Laboratório 1 - Computação Concorrente 2021-2

Ricardo Kaê - DRE 116 039 521

Atividade 1
Sim, há mudança na ordem de execução das threads. Dado duas execuções distintas do processo **hello**, a ordem de execução das threads no tempo são distintas. E devido a isso, diferentes resultados são impressos na tela.

1º execução	timeline
--Cria a thread 0	TP
--Cria a thread 1	TP
Hello World	TC
--Cria a thread 2	TP
Hello World	TC
--Cria a thread 3	TP
Hello World	TC
--Cria a thread 4	TP
Hello World	TC
--Cria a thread 5	TP
Hello World	TC
--Cria a thread 6	TP
Hello World	TC
--Cria a thread 7	TP
Hello World	TC
--Cria a thread 8	TP
Hello World	TC
--Cria a thread 9	TP
Hello World	TC
--Thread principal terminou	TP
Hello World	TC

2º execução	timeline
--Cria a thread 0	TP
--Cria a thread 1	TP
--Cria a thread 2	TP
Hello World	TC
Hello World	TC
Hello World	TC
--Cria a thread 3	TP
--Cria a thread 4	TP
Hello World	TC
--Cria a thread 5	TP
Hello World	TC
--Cria a thread 6	TP
Hello World	TC
--Cria a thread 7	TP
Hello World	TC
--Cria a thread 8	TP
Hello World	TC
--Cria a thread 9	TP
Hello World	TC
--Thread principal terminou	TP
Hello World	TC

Figura 1: Duas execuções aleatórias do processo **hello**

A razão para essa diferença, nas ordens de execução, está na política de escalonamento de threads do Sistema Operacional. O Sistema Operacional decide **qual, quando** e em **quanto tempo** cada thread ganhará de CPU e outros recursos de hardware para consumir.

No caso das duas execuções do processo **hello** ilustradas na Figura 1, podemos observar que no começo da 1º execução de **hello**, a thread principal (TP) ganhou um tempo de CPU mais curto do que na 2º execução. Assim, na primeira execução imprimiu menos vezes a string –**Cria a thread %d**, no começo do programa, do que na segunda execução. Consequentemente,

uma thread criada (TC) na 1º execução ganhou o acesso à CPU mais cedo do que na 2º execução, o que explica a diferença entre as ordens de impressão entre a primeira e a segunda execução. Ainda, pode ser notado na figura, outras mudanças na ordem das threads devido ao escalonamento do sistema em cada execução.

Atividade 2

A principal diferença em relação ao programa anterior é que agora podemos saber qual thread foi escalonada, pelo Sistema Operacional, para executar. Nesse programa, **hello_arg**, o ID da thread é passado como parâmetro para a função que a thread irá executar e então a mensagem a ser impressa conta com esse ID.

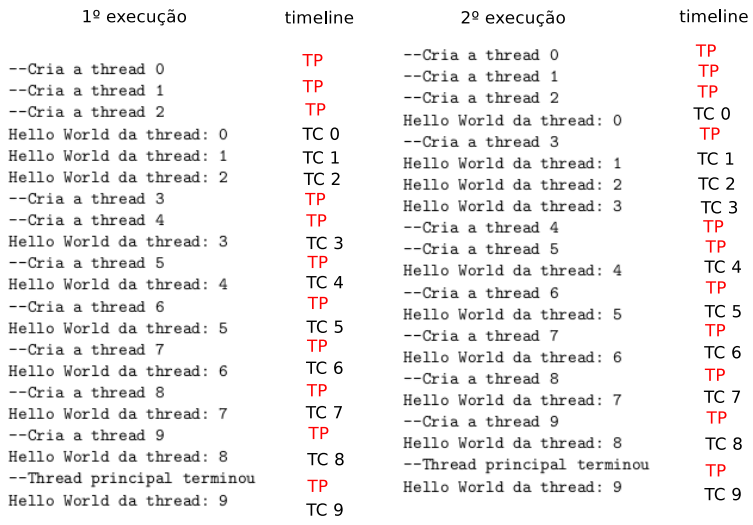

Figura 2: Duas execuções aleatórias do processo **hello_arg**

Devido ao fato do processamento ser igual e pouco significativo para todas as threads criadas (TC), isto é, todas apenas imprimem uma string. Quando escalonadas, elas finalizam imediatamente suas tarefas e como elas entram numa espécie de fila de execução assim que criadas, a ordem de execução das mensagens de impressão respeitam a ordem de criação das threads, que é feita segundo um ID crescente. E assim, mesmo em diferentes execuções do programa, a ordem crescente das threads criadas se mantém. Em todos os casos, primeiro executa a thread com ID 0, depois ID 1, depois ID 2 e assim por diante, até a thread de ID 9.

Contudo, ainda como no programa anterior, devido a política de escalonamento do Sistema Operacional, em cada execução do processo, há mudanças na ordem temporal de execução das threads. O que pode ser notado pela Figura 2, para duas execuções aleatórias do programa **hello_arg**.

Atividade 3
Sim, mais de um parâmetro é passado para função que a thread irá executar. E tais parâmetros são impressos na tela.

Atividade 4
A diferença é que devido ao comando **pthread_join**, a thread principal só finaliza a sua execução após todas as threads terem finalizado suas tarefas. O que é diferente das execuções anteriores em que, pelas Figuras 1 e 2, pode ser notado que a thread principal finaliza em certas execuções e ainda restam threads criadas que não haviam sido escalonadas. Assim, tais threads, nesses casos, emitem suas mensagens de impressão após a finalização da thread principal.

Com acréscimo do comando **phtread_join** para cada thread do sistema

```
//--espera todas as threads terminarem
  for (thread=0; thread<NTHREADS; thread++) {
    if (pthread_join(tid_sistema[thread], NULL)) {
         printf("--ERRO: pthread_join() \n"); exit(-1);
    }
  }

  printf("--Thread principal terminou\n");
```

A thread principal só finaliza após todas as threads criadas terem sido finalizadas. E isso, para todas as execuções do programa **hello_join**.

Atividade 5
Para a atividade 5, três threads são escalonadas: a thread principal e mais duas threads (0 e 1), que são construídas por essa principal para incrementar um vetor. A tarefa de incrementar o vetor é dividida entre essas duas threads da seguinte forma. A thread T0 (com ID 0) incrementa de 0 a 5000. E a thread T1 (com ID 1) incrementa de 5000 a 9999. E assim, as duas juntas percorrem um vetor de 10000 elementos, elevando ao quadrado cada

elemento. O código encontra-se abaixo e também no mesmo repositório do GitHub desse documento

```c
#include <stdio.h>
#include <pthread.h>

#define NTHREADS 2
#define TAM_V 10000
#define LIM_T1 5000

// Divide a tarefa de incrementar o vetor em dois seguimentos
// T1 incrementa de 0 a 5000 e T2 incrementa de 5000 a 9999
int vetor[TAM_V];
int t1_i = 0;
int t2_i = 5000;

void* pow2(void *arg)
{
    int tid = *(int*) arg;
    if (tid == 0)
        for (; t1_i < LIM_T1; t1_i++)
            vetor[t1_i] = vetor[t1_i] * vetor[t1_i];
    else
        for (; t2_i < TAM_V; t2_i++)
            vetor[t2_i] = vetor[t2_i] * vetor[t2_i];
    pthread_exit(NULL);
}

int main()
{
    int thread, i, tid_local[NTHREADS];
    pthread_t tid[NTHREADS];

    // 1. Inicializa o vetor
    for (i = 0; i < TAM_V; i++)
        vetor[i] = i;

    // 2. Cria as threads e passa seus ids para função pow2
    for (thread = 0; thread < NTHREADS; thread++) {
        tid_local[thread] = thread;
```

```
        if (pthread_create(&tid[thread], NULL, pow2, (void*) &tid_local[thread]))
            puts("Erro no pthread_create()");
    }

    // Espera as threads terminarem de elevar os elementos ao quadrado
    for (thread = 0; thread < NTHREADS; thread++) {
        if (pthread_join(tid[thread], NULL))
            puts("Erro no pthread_join()");
    }

    // Imprimindo vetor
    printf("# VETOR IMPRESSO #\ni \t i^2\n");
    for (i = 0; i < TAM_V; i++)
        printf("%d \t %d\n", i, vetor[i]);

    pthread_exit(NULL);
    return 0;


}
```